Research Article

Interlinking Developer Identities within and across Open Source Projects: The Linked Data Approach

Aftab Iqbal and Michael Hausenblas

Digital Enterprise Research Institute, National University of Ireland, Galway, Ireland

Correspondence should be addressed to Aftab Iqbal; aftab.iqbal@deri.org

Received 17 January 2013; Accepted 13 March 2013

Academic Editors: P. Ciancarini and M. Marzolla

Copyright © 2013 A. Iqbal and M. Hausenblas. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Software developers use various software repositories in order to interact with each other or to solve related problems. These repositories provide a rich source of information for a wide range of tasks. However, one issue to overcome in order to make this information useful is the identification and interlinking of multiple identities of developers. In this paper, we propose a Linked Data-based methodology to interlink and integrate multiple identities of a developer found in different software repositories of a project as well as across repositories of multiple projects. Providing such interlinking will enable us to keep track of a developer’s activity not only within a single project but also across multiple projects. The methodology will be presented in general and applied to 5 Apache projects as a case study. Further, we show that the few methods suggested so far are not always appropriate to overcome the developer identification problem.

1. Introduction and Motivation

In Software Engineering, many tools with underlying repositories have been introduced to support the collaboration and coordination in distributed software development. Research has shown that these software repositories contain rich amount of information about software projects. By mining the information contained in these software repositories, practitioners can depend less on their experience and more on the historical data [1]. However, software repositories are commonly used only as record-keeping repositories and rarely for design decision processes [2]. Examples of software repositories are [3] source control repositories, bug repositories, archived communication, and so forth.

Developers (we will use the term “developer” to represent the core developers, contributors, bug reporters and users of an open source project) use these repositories to interact with each other or to solve software-related problems. By extracting rich information from these repositories, one can guide decision processes in modern software development. For example, source code and bugs are quite often discussed on bug repositories and project mailing lists. Data in these software repositories could be analyzed to extract bug and source code related discussions, which could be linked to the actual bug description and source code. This could allow keeping track of developers discussion related to a bug or source code in different software repositories.

Developers are required to adopt an identity for each software repository they want to use. For example, they are required to adopt an email address in order to send an email to the project mailing list, adopt an ID (we will use the term “ID” to represent different types of identities which developers use to interact with software repositories), push commits to the source control repository, adopt an ID to report bugs or post comments on a bug in bug repository, and so forth. Often developers adopt different IDs for each software repository and sometimes multiple IDs for the same repository [4], while interacting with these software repositories in different context. The software repositories are designed to help developers collaborate and coordinate but lack the feature to manage and merge multiple IDs of a developer specially in the case of mailing list and bug tracking repositories. Hence, a developer can register multiple IDs and use them to collaborate and coordinate with other developers.
For example, Figure 1 shows a subset of the social network graph derived from the communication occurred among developers on the mailing list of an Apache project.

In the figure, we colored two nodes (i.e., red and blue) showing that both email ID(s) belong to a developer X (we shortened the labels of each node in order to keep the privacy of developers). It is quite clear that the developer X has used both email ID(s) to communicate with other developers on the mailing list. Looking into the social structure of graphs (based on both ID(s)), we see that there are few ID(s) which are common in both graphs but most of the ID(s) are part of different social graphs of developer X. In an ideal scenario, either both social graphs need to be merged into one graph or at least there should exists some sort of information conveying that both ID(s) belong to developer X. Without any information provided about the different IDs used by a developer, a researcher who carries out social network analysis research [5–7] might consider both ID(s) as two different developers because using distinct ID(s) makes developers appear as different entities. Furthermore, analysis of developer’s activity [8] within a project might be difficult if the developer is using different and multiple IDs for each software repository.

In the specific case of open source software development, developers are often involved in multiple projects. One of the many reasons is that they see their contribution of time and efforts as a gift they provide to the project community. Other reasons could be a developer’s ability to acquire skills and experience by participating in multiple open source projects. Hence the development activity of a developer is distributed not only within different software repositories of a project but also across different projects. It is worth mentioning that there exists an implicit connection between developer’s activity within different software repositories of a project and also across different projects. Therefore, we require methods and techniques to correctly identify and relate the different IDs of developers not only within a project but also across different projects.

The contribution of this paper is manifold: we propose a simple yet effective Linked Data-based approach to identify
and interlink multiple IDs of a developer within different software repositories of a project as well as across software repositories of different projects. We show how interlinking helps in integrating developer’s development activity which is distributed within different software repositories of a project due to the usage of multiple IDs. Further, we show how the interlinking enables us to keep track of developer’s activity across different projects. Further, we compare the matching results of our approach with other existing developer identification approaches.

The paper is structured as follows. In Section 2 we review related work and compare it to our approach. Research questions are outlined in Section 3. In Section 4, we present our approach to identify and interlink multiple IDs of a developer. We report our evaluation on 5 Apache projects in Section 5 followed by the discussion on the results and research questions in Section 6. Finally, we conclude and outline future steps in Section 7.

2. Related Work

To the best of our knowledge, there are only a few published works on identifying and relating the different IDs that developers use to interact with software repositories of a project. In [9], Bird et al. proposed an approach to produce a list of <name, email> pairs by parsing the emails and clustering them. The clustering algorithm to measure the similarity between every pair of ID(s) is based on string similarity between names, between emails, between names and emails, and so forth. Two ID(s) with a similarity measure lying below a predefined threshold value are placed into the same cluster. The authors use different approaches to compute the similarity measures between every pair of ID(s), some of which we tested on our data set to validate the effectiveness of these approaches (cf. Section 5). We also use the string similarity approach in order to interlink the ID(s) of a developer but our scope is broader in a sense that we are applying the heuristics not only within a single project or a repository but also across repositories of multiple projects.

In [4], Robles and Gonzalez-Barahona discuss the problem of developer identification in general, but the work lacks details about the heuristics they propose to identify and match the different IDs of developers. This makes it difficult to validate their approaches for solving this problem. The authors propose a technique to build one identity from another by extracting the “real life” name from email addresses, such as surname@domain.com, name.surname@domain.com, and so forth. This is an approach based on predefined name schemes. Further, they try to match user names obtained from CVS to email addresses (excluding the domain after “@”). Their approach also relies on string similarity algorithm.

In [10], Conklin presents a methodology to integrate data about projects by way of matching projects (entities) based on certain heuristics. The heuristics include a simple scoring system for confidence in pairwise project matches. The author proposes to interlink similar projects which are listed on multiple code forges. On the contrary, in this paper we focused on identification and interlinking multiple IDs of a developer.

In general, the problem is related to duplicate detection. The duplicate detection frameworks provide several techniques to effectively solve matching different entities. In this regard, Kopcke and Rah [11] analyzed and compared 11 different duplicate detection frameworks. While research in this area mostly refers to identifying duplicates in the same data set, the techniques might be mapped to the case of matching over different data sets. However, they are tailor-made for identifying different IDs of the same developer inside one repository. Naumann and Herschel [12] provides a nice overview of this research direction.

In the Semantic Web domain, Volz et al. [13] proposed an interlinking framework also known as SILK framework which generates link between two RDF data sets based on some string similarity measures specified by the user. Their interlinking framework supports different string similarity algorithms to compare if two different RDF resources are similar or not. In a next step, we will assess to what extent we can use the SILK framework to interlink developer identities in the near future.

3. Research Questions

In the following, we list down few research questions which will be addressed later (Section 6) based on interlinking multiple IDs of a developer.

(1) RQ-1: What is the added benefits of interlinking multiple IDs of a developer? We will investigate if developers are using multiple IDs within a software repository and if interlinking them will provide any added benefits. We will validate this on the mailing list data set by computing the distinct nodes which are not shared among different social graphs of a developer using multiple IDs. Further, we will compute the number of bugs reported by a developer using multiple IDs. This would show us a clear picture on the amount of information which was not previously connected.

(2) RQ-2: What is the ratio of developer’s existence in multiple projects? We will investigate the frequency of developer’s participation in multiple projects. We will evaluate if developer’s participation follow Power’s law that is, many developers participate in few projects and only few developers participate in many projects.

(3) RQ-3: What are the contributions made by developers in multiple projects? As mentioned in Section 1, developers often contribute to multiple open source projects. Therefore, we will evaluate to what extent our interlinking approach helps us in tracing developer’s activity across multiple projects. In particular, we will investigate if developers are highly active in pushing source control commits, reporting bugs, and participating in discussions across multiple projects.
4. Our Approach

In this section, we describe our approach for identifying and interlinking different IDs of a developer found in different software repositories of a project as well as across repositories of multiple projects. We considered mailing list and bug repository data sources in this paper because developers often use multiple IDs on these software repositories as opposed to source control repository where access and accounts are controlled centrally.

With “Linked Data Driven Software Development” (LD2SD) [14], we have introduced a Linked Data-based methodology to relate and integrate data across software repositories explicitly and unambiguously. We propose to use Semantic Web technologies to represent data from different software repositories. As such, we propose to use (Resource Description Framework) [15] RDF as the core, target data model. Once modeled in RDF, the data can be easily integrated, indexed and queried using the SPARQL query (http://www.w3.org/TR/rdf-sparql-query/) standard and associated tools. Finally, the integrated data can be published on the Web using Linked Data principles (http://www.w3.org/DesignIssues/LinkedData.html) allowing third parties to discover and subsequently crawl the knowledge, and also allowing to interlink with background or other related information available remotely on the Web. We refer the readers to [16] for details on how these standards would be used. Instead here we focus on the identification and interlinking of developer ID(s) in software repositories.

As mentioned by Bird et al., most emails contain header of the form [9]

From: "aftab iqbal" <ai@example.com>

For each email, we represent header information in RDF using FOAF (http://www.foaf-project.org/) ontology as shown in Listing 1 (lines 10–12) (the URIs used in the listings are for illustration purpose only and are not dereferenceable). We used email alias to build developer URI (i.e., a resource in RDF) because email alias can be used to distinguish between different developers. Further, the name and email address are defined as property values of that resource.

While converting mailing list data to RDF using our custom written scripts, we found certain developers who have used either full or part of their names in the email header. For example

From: "muhammad aftab iqbal" <ai@example.com>

The above two email headers reveal multiple name(s) of the sender (i.e., aftab iqbal and muhammad aftab iqbal). The developer URI will remain the same in the case of the above two exemplary email headers (i.e., http://srvgal85.deri.ie/linkedfloss/ai) but the property values will be changed (name in that case) as each email header belongs to a different email. Therefore, multiple name(s) can be easily extracted to produce a list of <name, id> pairs for a particular developer. Each bug usually represents developer information of the form

<assignee username="aftab iqbal">aftab iqbal</assignee>

<reporter username="jim">jim</reporter>

We represent developer information for each bug in RDF using FOAF ontology as shown in Listing 2 (lines 10–12).

After transforming the data into RDF, we loaded the RDF data sets into SPARQL endpoint (http://linkedfloss.srvgal85.deri.ie/sparql). The next step is to identify and interlink multiple IDs of a developer. From the Listings 1 and 2, we are able to conclude that the developer (aftab iqbal) has used different IDs (i.e., ai and aftab iqbal) while interacting with others on the mailing list and bug tracking repository. We can interlink these two RDF fragments as shown in Listing 3 using an owl:sameAs property indicating that the developer URLs actually refer to the same entity.

By explicitly interconnecting ID(s) of a developer, we will be able to trace developer’s activity on the mailing list as well as bug tracking repository. However, manually identifying and interlinking ID(s) of a developer is a time consuming task especially when dealing with large amount of data. Hence, one is required to devise an automatic approach for the identification and interlinking of developer ID(s). Therefore, in the following we will explain our simple yet effective developer matching approach.

---

LISTING 1: An exemplary email RDFication.

```xml
@prefix foaf: <http://xmlns.com/foaf/0.1/>.
@prefix sioc: <http://rdfs.org/sioc/types/>.
@prefix email: <http://simile.mit.edu/2005/06/ontologies/email/>.
@prefix : <http://srvgal85.deri.ie/linkedfloss/email2rdf/>.
:s10029029 a sioc:MailMessage;
email:subject "compilation problem";
email:body "while compiling the latest source i am ..."
.
<http://srvgal85.deri.ie/linkedfloss/ai> a foaf:Person;
foaf:name "aftab iqbal";
foaf:mbox <mailto:ai@example.com>.
...
```

For each email, we represent header information in RDF using FOAF (http://www.foaf-project.org/) ontology as shown in Listing 1 (lines 10–12) (the URIs used in the listings are for illustration purpose only and are not dereferenceable). We used email alias to build developer URI (i.e., a resource in RDF) because email alias can be used to distinguish between different developers. Further, the name and email address are defined as property values of that resource.

While converting mailing list data to RDF using our custom written scripts, we found certain developers who have used either full or part of their names in the email header. For example

From: "muhammad aftab iqbal" <ai@example.com>

The above two email headers reveal multiple name(s) of the sender (i.e., aftab iqbal and muhammad aftab iqbal). The developer URI will remain the same in the case of the above two exemplary email headers (i.e., http://srvgal85.deri.ie/linkedfloss/ai) but the property values will be changed (name in that case) as each email header belongs to a different email. Therefore, multiple name(s) can be easily extracted to produce a list of <name, id> pairs for a particular developer. Each bug usually represents developer information of the form

<assignee username="aftab iqbal">aftab iqbal</assignee>

<reporter username="jim">jim</reporter>

We represent developer information for each bug in RDF using FOAF ontology as shown in Listing 2 (lines 10–12).

After transforming the data into RDF, we loaded the RDF data sets into SPARQL endpoint (http://linkedfloss.srvgal85.deri.ie/sparql). The next step is to identify and interlink multiple IDs of a developer. From the Listings 1 and 2, we are able to conclude that the developer (aftab iqbal) has used different IDs (i.e., ai and aftab iqbal) while interacting with others on the mailing list and bug tracking repository. We can interlink these two RDF fragments as shown in Listing 3 using an owl:sameAs property indicating that the developer URLs actually refer to the same entity.

By explicitly interconnecting ID(s) of a developer, we will be able to trace developer’s activity on the mailing list as well as bug tracking repository. However, manually identifying and interlinking ID(s) of a developer is a time consuming task especially when dealing with large amount of data. Hence, one is required to devise an automatic approach for the identification and interlinking of developer ID(s). Therefore, in the following we will explain our simple yet effective developer matching approach.
4.1. Preliminaries. Let Authors be a list of distinct developer URIs found in the data set:

Authors = \{A, B, C, D, \ldots\}, \quad (1)

where

\[ A = \text{http://srvgal85.deri.ie/linkedfloss/ai}, \]
\[ B = \text{http://srvgal85.deri.ie/linkedfloss/aftab.iqbal}. \]

Let \( \hat{A} \) and \( \hat{B} \) be the distinct <name, id> pairs for A and B respectively

\[
\hat{A} = \left( <\text{aftab iqbal, ai@example.com}> <\text{muhammad aftab iqbal, ai@example.com}> \right), \\
\hat{B} = \left( <\text{aftab iqbal, aftab.iqbal@example.com}> \right). 
\]

(3)

Let \( L \) be a set which contains a collection of distinct developer URIs along with their associated <name, email> pairs

\[
L = \left( \begin{array}{c}
(A, \hat{A}) \\
(B, \hat{B}) \\
(C, \hat{C}) \\
\vdots
\end{array} \right). \quad (4)
\]

4.2. Interlinking ID(s) within a Repository. We compared each developer URI with every other developer URI in set \( L \) by comparing their respective <name, id> (where id means excluding everything after “@”) pairs. If the match is found then we established an \( \text{owl:sameAs} \) link between both developer URIs as shown in Listing 3. The pseudocode of our approach is shown in Algorithm 1.

4.3. Interlinking ID(s) across Projects. Let \( \text{Prj}_A \) and \( \text{Prj}_B \) (cf. (5)) be the sets which contain the collection of distinct developer URIs along with their associated <name, email> pairs extracted from the repositories of project A and project B such that

\[
\text{Prj}_A = \left( \begin{array}{c}
(A, \hat{A}) \\
(B, \hat{B}) \\
(C, \hat{C}) \\
\vdots
\end{array} \right), \\
\text{Prj}_B = \left( \begin{array}{c}
(A, \hat{A}) \\
(B, \hat{B}) \\
(C, \hat{C}) \\
\vdots
\end{array} \right). \quad (5)
\]

For each developer URI, we queried the \( \text{owl:sameAs} \) links (if any) in order to build a comprehensive list of <name, email> pairs. Later, we compared each developer URI in set \( \text{Prj}_A \) with every developer URI in set \( \text{Prj}_B \) by comparing their respective <name, email> pairs. If the match is found then we established an \( \text{owl:sameAs} \) link between both developer URIs (cf. Listing 3). The pseudocode of our approach is shown in Algorithm 2.
ISRNSoftwareEngineering

\[ \text{lstURI} \leftarrow \text{get distinct developer URIs} \]
\[ \text{for } i \leftarrow 0 \text{ to lstURI.size()} \]
\[ \quad \text{do} \]
\[ \quad \quad \text{lst} \leftarrow \text{id(s) and name(s) associated with lstURI[i]} \]
\[ \quad \quad \text{hshMap : add(devURI[i], lst)} \]
\[ \text{comment: match each developer URI with every other developer URI in} \]
\[ \text{comment: the list based on the <name,id> pairs} \]
\[ \text{for } i \leftarrow 0 \text{ to hshMap.size()} \]
\[ \quad \text{id1} \leftarrow \text{hshMap.getKey(i)} \]
\[ \quad \text{lst1} \leftarrow \text{hshMap.getValue(id1)} \]
\[ \quad \text{for } j \leftarrow i + 1 \text{ to hshMap.size()} \]
\[ \quad \quad \text{do} \]
\[ \quad \quad \quad \text{id2} \leftarrow \text{hshMap.getKey(j)} \]
\[ \quad \quad \quad \text{lst2} \leftarrow \text{hshMap.getValue(id2)} \]
\[ \quad \quad \quad \text{if matched(lst1, lst2)} \]
\[ \quad \quad \quad \quad \text{then } \text{generateSameAsLink(id1, id2)} \]

Algorithm 1: ID_LINKING_IN_A_REPOPOSITORY().

\[ \text{prj}_A \leftarrow \text{get distinct developer URIs from project A} \]
\[ \text{prj}_B \leftarrow \text{get distinct developer URIs from project B} \]
\[ \text{for } i \leftarrow 0 \text{ to prj}_A.size() \]
\[ \quad \text{lstID} \leftarrow \text{getSameAsLinks(prj}_A[i]) \]
\[ \quad \text{for } j \leftarrow 0 \text{ to lstID.size()} \]
\[ \quad \quad \text{do} \]
\[ \quad \quad \quad \text{lst} \leftarrow \text{get email(s) and name(s) associated with lstID[j]} \]
\[ \quad \quad \quad \text{lstPairs.add(lst)} \]
\[ \quad \quad \quad \text{hshMap}_A.add(prj}_A[i], \text{lstPairs)} \]
\[ \text{for } i \leftarrow 0 \text{ to prj}_B.size() \]
\[ \quad \text{lstID} \leftarrow \text{getSameAsLinks(prj}_B[i]) \]
\[ \quad \text{for } j \leftarrow 0 \text{ to lstID.size()} \]
\[ \quad \quad \text{do} \]
\[ \quad \quad \quad \text{lst} \leftarrow \text{get email(s) and name(s) associated with lstID[j]} \]
\[ \quad \quad \quad \text{lstPairs.add(lst)} \]
\[ \quad \quad \quad \text{hshMap}_B.add(prj}_B[i], \text{lstPairs)} \]
\[ \text{comment: match each developer URI of project A with every developer URI} \]
\[ \text{comment: in project B based on the <name,email> pairs} \]
\[ \text{for } i \leftarrow 0 \text{ to hshMap}_A.size() \]
\[ \quad \text{id1} \leftarrow \text{hshMap}_A.getKey(i)} \]
\[ \quad \text{lst1} \leftarrow \text{hshMap}_A.getValue(id1)} \]
\[ \quad \text{for } j \leftarrow 0 \text{ to hshMap}_B.size() \]
\[ \quad \quad \text{do} \]
\[ \quad \quad \quad \text{id2} \leftarrow \text{hshMap}_B.getKey(j)} \]
\[ \quad \quad \quad \text{lst2} \leftarrow \text{hshMap}_B.getValue(id2)} \]
\[ \quad \quad \quad \text{if matched(lst1, lst2)} \]
\[ \quad \quad \quad \quad \text{then } \text{generateSameAsLink(id1, id2)} \]

Algorithm 2: ID_LINKING_ACROSS_PROJECTS().

The outcome of our above described matching approach(es) is a set of RDF files in N-TRIPLES (http://www.w3.org/2001/sw/RDFCore/ntriples/) format which contains the developer URIs having owl:sameAs links between them (e.g., see Listing 3). The RDF files were later loaded into the SPARQL endpoint so that it can be used to query development activity of a developer within and across projects.

5. Evaluation
Before we discuss the results of our matching approach(es), we describe the Apache projects selected for evaluation. We gathered data from software repositories of 5 Apache projects (c.f Table 1). The reason of choosing Apache projects is that the repositories of these projects are on the Web and are available to download (i.e., mailing list archives, bugs, etc.).
We selected data from the beginning of each Apache project to date as listed in Table 1.

5.1. Interlinking ID(s) within a Repository. We applied our interlinking approach (cf. Algorithm 1) on the mailing list and bug tracking repositories separately in order to identify and interlink multiple IDs of a developer within a repository. During the matching phase, we found certain IDs which were more generic and multiple developers in each project were associated to those IDs. We excluded those ID(s) during the matching phase. Example of those ID(s) are jakarta, jakarta-ant, lists, ant-dev, dev, ant, apache, general, hadoop, nutch-dev, log4j, log4j-dev, java-dev, lucene, and so forth. The outcome of our interlinking approach is listed in Tables 2 and 3 where the ”links” column tells the total number of owl:sameAs links established between ID(s), and the ”ID(s) found” column tells the distinct ID(s) found with a match.

From Tables 2 and 3, it is quite obvious that the usage of multiple IDs is mostly common on the mailing list than the bug tracking repository except the special case of Apache ANT project. However, it is still important to identify and interlink multiple IDs of a developer in every software repository. The same interlinking approach can be used to match developer ID(s) across different repositories (i.e., matching mailing list and bug tracking repository developer ID(s)) of a project.

5.2. Interlinking ID(s) across Projects. In order to identify and interlink ID(s) of a developer across different projects, we applied our interlinking approach (cf. Algorithm 2) to the bug tracking and mailing list repositories of the Apache projects separately. We considered two ID(s) to be similar if either the <name, email> pairs of both ID(s) matched or if only the name or email address of the ID(s) matched. During the evaluation, we found match for certain developers who have used single word name (e.g., chris, brian, j, etc.). As it is difficult to differentiate if the developers are the same (based on a single word name), we only considered a match if the developer name consists of at-least two words (i.e., name and surname). The results of our cross project matching approach based on mailing list and bug tracking repositories are shown in Tables 4 and 5.

The results show that matching based on either name or email address generates more links comparing to the <name, email> approach. Also, there are good number of developers whose ID(s) are found in multiple projects. The most prominent among them are the developers of Apache ANT and Apache Maven project. The reasons behind it could be the technical dependencies between both projects (e.g., using project components) or the interest of developers working on similar kind/category of projects.

5.3. Comparison of Developer Identification Approaches. In this section, we apply few methods suggested so far (see Section 2) on our data set in order to validate the effectiveness of these methods comparing to our approach. We applied the
Table 5: Matching bug repository ID(s) across projects.

<table>
<thead>
<tr>
<th>Project A (no. of developers)</th>
<th>Project B (no. of developers)</th>
<th>Bug tracking system</th>
<th>ID(s) found</th>
<th>Links</th>
<th>ID(s) found</th>
<th>Links</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apache Ant (3927)</td>
<td>Apache Hadoop (1219)</td>
<td></td>
<td>20</td>
<td>10</td>
<td>68</td>
<td>35</td>
</tr>
<tr>
<td>Apache Ant (3927)</td>
<td>Apache Logging (1131)</td>
<td></td>
<td>242</td>
<td>121</td>
<td>363</td>
<td>197</td>
</tr>
<tr>
<td>Apache Ant (3927)</td>
<td>Apache Lucene (981)</td>
<td></td>
<td>24</td>
<td>12</td>
<td>61</td>
<td>34</td>
</tr>
<tr>
<td>Apache Ant (3927)</td>
<td>Apache Maven (1577)</td>
<td></td>
<td>74</td>
<td>37</td>
<td>261</td>
<td>143</td>
</tr>
<tr>
<td>Apache Hadoop (1219)</td>
<td>Apache Logging (1131)</td>
<td></td>
<td>2</td>
<td>1</td>
<td>19</td>
<td>10</td>
</tr>
<tr>
<td>Apache Hadoop (1219)</td>
<td>Apache Lucene (981)</td>
<td></td>
<td>44</td>
<td>22</td>
<td>45</td>
<td>23</td>
</tr>
<tr>
<td>Apache Hadoop (1219)</td>
<td>Apache Maven (1577)</td>
<td></td>
<td>20</td>
<td>10</td>
<td>20</td>
<td>10</td>
</tr>
<tr>
<td>Apache Logging (1131)</td>
<td>Apache Lucene (981)</td>
<td></td>
<td>4</td>
<td>2</td>
<td>16</td>
<td>8</td>
</tr>
<tr>
<td>Apache Logging (1131)</td>
<td>Apache Maven (1577)</td>
<td></td>
<td>32</td>
<td>16</td>
<td>98</td>
<td>52</td>
</tr>
<tr>
<td>Apache Lucene (981)</td>
<td>Apache Maven (1577)</td>
<td></td>
<td>16</td>
<td>8</td>
<td>27</td>
<td>14</td>
</tr>
</tbody>
</table>

In this section, we discuss the research questions (cf. Section 3) based on the results we achieved through interlinking multiple IDs of a developer.

RQ-1: What is the added benefits of interlinking multiple IDs of a developer?

In order to evaluate the added benefits of interlinking multiple IDs of a developer, we calculated the difference in the social graphs constructed from the mailing list data set and the number of bugs reported by a developer whose multiple IDs were not interlinked. We wanted to find out how much of the developer’s development activity is not explicitly interconnected due to the usage of multiple IDs within a software repository.

We calculated the difference between social graphs of a developer who has multiple ID(s) for each Apache project. The social graphs were constructed by analyzing the reply structure of the email threads. The difference between two social graphs belonging to the same developer was calculated by computing the number of nodes (i.e., ID) which are part of one social graph of a developer and are missing in the other social graph of that particular developer. This would reveal distinct nodes which are part of different social graphs of a developer and lack a common connection (as motivated in Figure 1). We sorted the results and plotted it in the form of a chart as shown in Figure 3.
The charts plotted show a potential difference in the social graphs of a developer who has multiple IDs. For example, in Apache Maven project (cf. Figure 3), we found a developer whose multiple social graphs had a difference of 613 nodes. This means that there are 613 nodes which are not shared among his multiple social graphs. We further computed the number of bugs reported by a developer using multiple IDs and plotted the results in the form of chart as shown in Figure 4. We stacked up the bugs reported by a developer using multiple IDs to show the total number of bugs he/she reported.

The results depicted in Figures 3 and 4 show that establishing owl:sameAs links between multiple IDs of a developer helps in better integration of data sources from the research (i.e., carrying out social network analysis) as well as the software development point of view (i.e., tracing developer activity in a repository). However, based on the small number of Apache projects we selected for this study, the bugs reporting activity using multiple IDs is not significant comparing to the mailing list’s social graph differences but we believe that one might find significant differences if it is applied to a large number of open source projects.

RQ-2: What is the ratio of developer’s existence in multiple projects?

Based on the evaluation data from 5 Apache projects, we queried owl:sameAs links of each developer across different Apache projects under consideration. Later, we counted the number of projects for which a developer has an owl:sameAs link and plotted the results as shown in Figure 5.

The chart shows that developer’s participation in multiple projects follow Power’s law because of the large number of developer’s participation in 2 Apache projects in contrast to participation in 3, 4, or 5 Apache projects. For each Apache project, we found a small subset of developers participating in at-least 4 Apache projects. It shows us a great tendency of developer’s participation in multiple projects. It further opens up new research challenges in order to access, evaluate, and analyze developer’s behavior while contributing to multiple projects in a given period of time.

RQ-3: What are the contributions made by developers in multiple projects?

In order to trace developer’s activity across multiple projects, we queried and retrieved owl:sameAs links of each developer. Later, we computed the development activity of only those developers who participated in multiple projects. In Figure 6, we have summarized the development activity of few developers who contributed to or participated in at-least 2 Apache projects.
The figure shows that only few developers have source control commit rights on multiple projects while most developers have commit rights on a single project. Further, the results show that developers are mostly active in exchanging emails and participating in bugs related discussions comparing to bugs reporting while contributing to multiple projects. There is a possibility that the developers contributed to the source code of multiple projects by submitting source code patches to the mailing list or bug tracking repository. We opt out this particular study as it requires interlinking source code authorship to the developers which is not in the current scope of this paper.

Based on the results, we believe that if the same methodology is applied to a large number of open source projects then we will see a significant amount of developer's activity across multiple projects.

7. Conclusion

We have motivated and proposed a simple yet effective approach to identify and interlink developer ID(s) within different software repositories of a project as well as across different projects. We have shown that explicitly establishing the interconnection (i.e., Linked Data approach) between multiple IDs of a developer helps in integrating, querying, and tracking developer activities in software repositories of a single project as well as across multiple projects.

We plan to combine existing developer identification methods with our methodology which might help in improving the interlinking approach, yielding higher quality and quantity links for developer ID(s). We further plan to integrate other potential developer-related open source repositories (i.e., analytical services like Ohloh, code forges, etc.) as discussed elsewhere [23]. We also plan to include more Apache projects and establish cross-project interlinking among developer ID(s). Based on that, we will study the social behavior of developers who had contributed to different projects in a given period of time. We believe that this will give new insights into the social and technical aspects of open source software developers.
Acknowledgment

The work presented in this paper has been funded by Science Foundation Ireland under Grant no. SFI/08/CE/I1380 (Lion-2).

References

